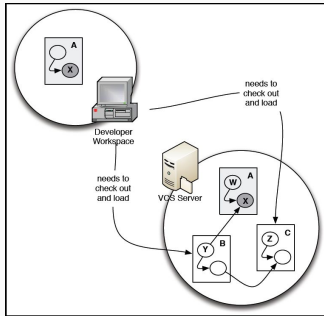


Taming Large Models with Hawk and NeoEMF

A. García-Domínguez, D. S. Kolovos, K. Barmpis, G. Daniel, G. Sunyé
MoDELS'2018, 14–19 October 2018

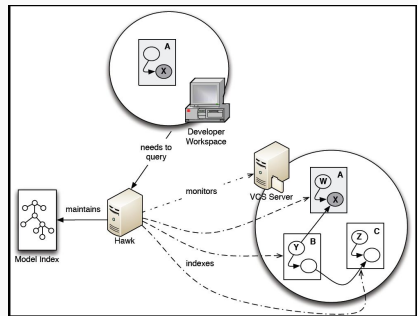
Hawk

Hawk: indexing for fast querying over fragment collections



Usual approach

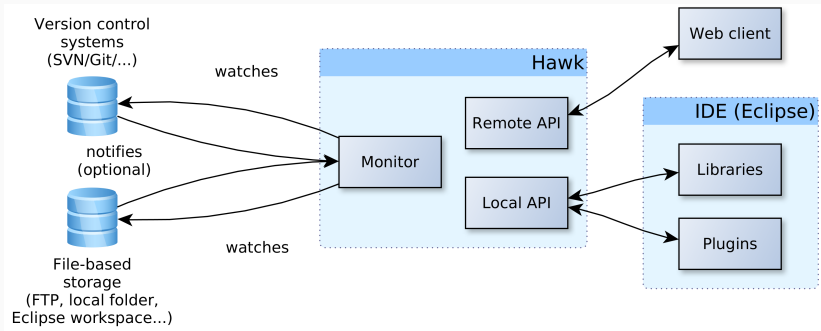
1. Check out **all** files from VCS
2. Load fragments into memory
3. Run query (might go over all fragments)



With Hawk

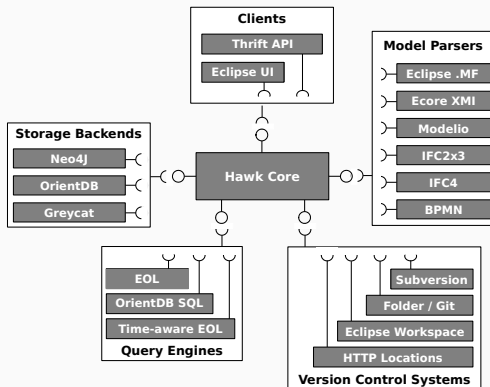
1. Hawk watches VCS, indexes
2. User queries Hawk over WS
3. Hawk runs query through NoSQL database efficiently
4. Hawk replies with result

Deployment



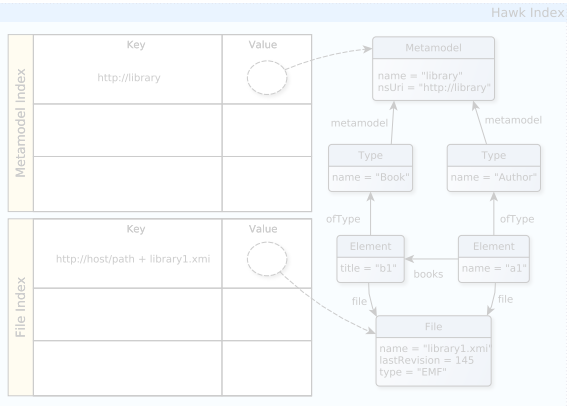
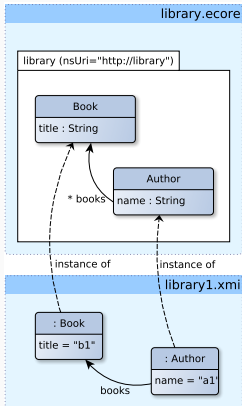
- Hawk can run as Eclipse plug-in, Java library, or network service
- We can have it watch over various types of locations:
 - Version control systems (SVN/Git repositories)
 - File stores (local folders, Eclipse workspaces, HTTP locations)

Component-based architecture



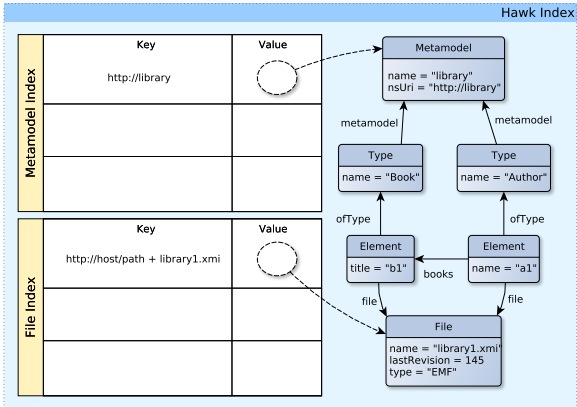
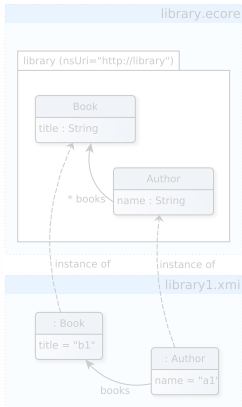
- Core: incremental graph updating + component interfaces
- Backends: Neo4j (fastest), OrientDB (multi-master), Greycat
- Clients: Eclipse GUI, cross-language Apache Thrift web services
- Query engines: Epsilon Object/Pattern Languages, OrientDB SQL
- Model parsers: EMF/Modelio models, Eclipse plug-in manifests...

Example for a library model



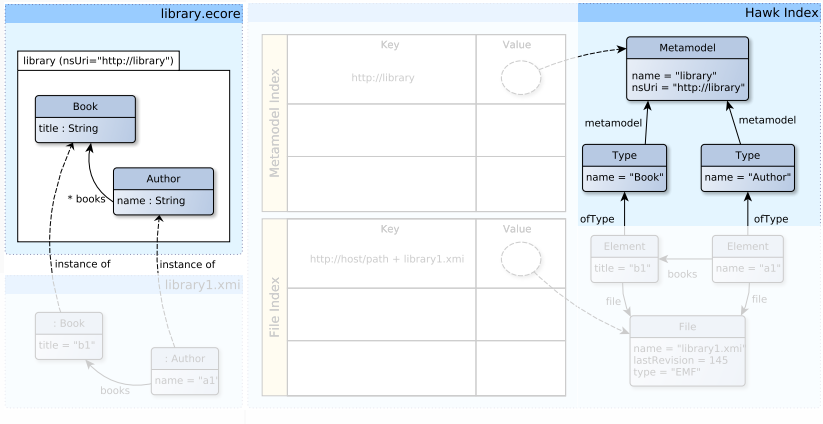
We go from these model files...

Example for a library model



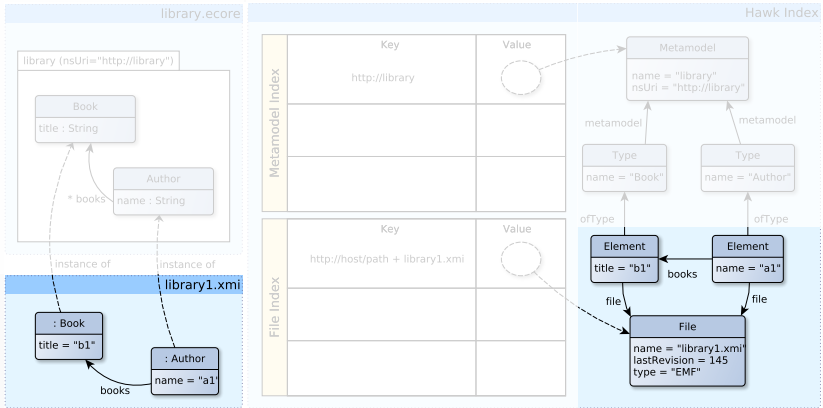
... to these NoSQL graphs.

Example for a library model



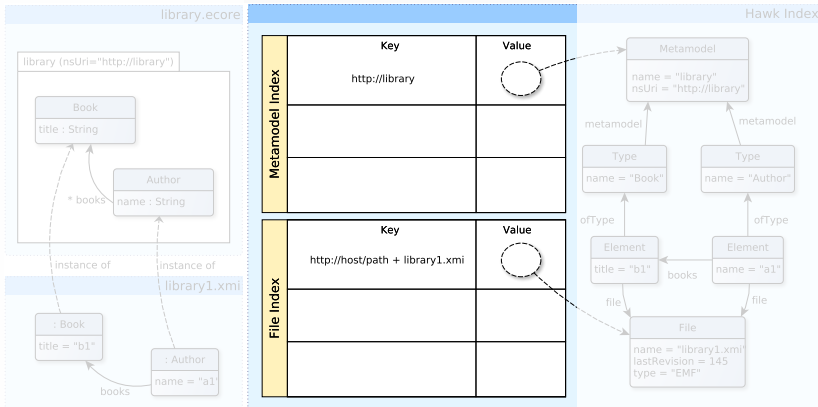
- Ecore packages → metamodel nodes
- Ecore classes → type nodes

Example for a library model



- Physical files → file nodes
- Model elements → element nodes

Example for a library model

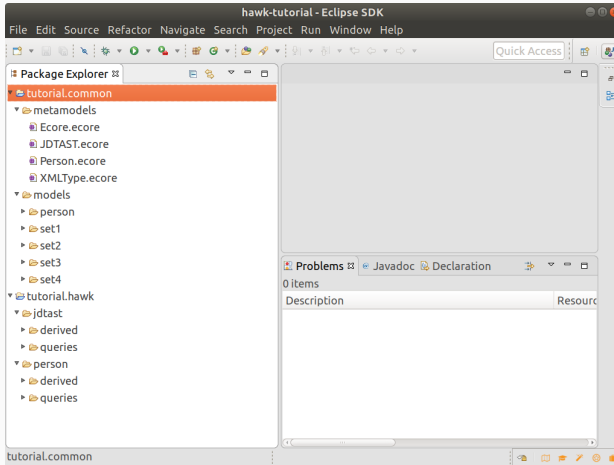


- MM index: package URI → metamodel node
- File index: file path → file node
- Users can define custom indices by attribute/expression

Hands-on time!

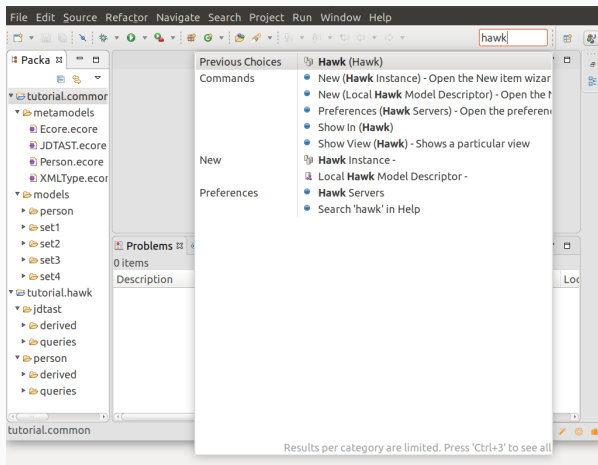
Let's index a Java model and find
singletons.

Preparing the workspace

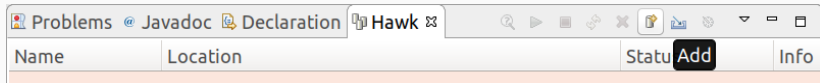


Open Eclipse and import `tutorial.common` and `tutorial.hawk`.

Opening the main Hawk view



Adding a new Hawk index



Click on the “Add” button. We can keep as many different indices as we want within the same Eclipse installation.

Options, options!

The screenshot shows a 'New Hawk Instance' wizard window. At the top, it says 'This wizard creates a new Hawk Instance to index model repositories.' Below this, there are several configuration fields: 'Name' (myhawk), 'Instance type' (org.hawk.core.runtime.LocalHawkFactory), 'Local storage folder' (/home/antonio/workspaces/hawk-tutorial/myhawk), 'Remote location' (http://localhost:8080/thrift/hawk/tuple), 'Enabled plugins' (a list of 10 plugins with checkboxes), 'Back-end' (org.hawk.neo4j_v2.Neo4JDatabase), and 'Min/Max Delay' (5000 and 512000). At the bottom, there are 'Cancel' and 'Finish' buttons.

New Hawk Instance

This wizard creates a new Hawk Instance to index model repositories.

Name:

Instance type:

Local storage folder:

Remote location:

Enabled plugins:

- ☒ org.hawk.bpmn.metamodel.BPMNMetaModelResourceFactory
- ☒ org.hawk.bpmn.model.BPMNModelResourceFactory
- ☒ org.hawk.emf.metamodel.EMFMetaModelResourceFactory
- ☒ org.hawk.emf.model.EMFModelResourceFactory
- ☒ org.hawk.graph.updater.GraphModelUpdater
- ☒ org.hawk.modelio.xml.listeners.ModelioGraphChangeListener
- ☒ org.hawk.modelio.xml.metamodel.ModelioMetaModelResourceFactory
- ☒ org.hawk.modelio.xml.model.ModelioModelResourceFactory
- ☒ org.hawk.timeaware.graph.TimeAwareModelUpdater
- ☒ org.hawk.uml.metamodel.UMLMetaModelResourceFactory
- ☒ org.hawk.uml.model.UMLModelResourceFactory

Back-end:

Min/Max Delay:

There are many options here: we can use a local or a remote Hawk, we can choose the backend, and we can enable/disable plugins at will.

Options for the JDTAST models

New Hawk Instance

This wizard creates a new Hawk Instance to index model repositories.

Name:

Instance type:

Local storage folder:

Remote location:

Enabled plugins:

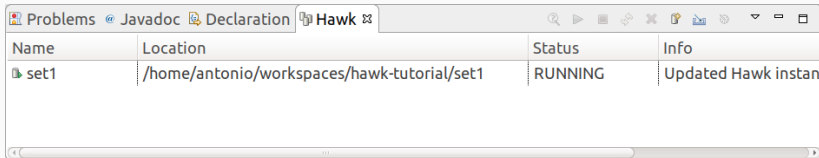
- ☐ org.hawk.bpmn.metamodel.BPMNMetaModelResourceFactory
- ☐ org.hawk.bpmn.model.BPMNModelResourceFactory
- ☒ org.hawk.emf.metamodel.EMFMetaModelResourceFactory
- ☒ org.hawk.emf.model.EMFModelResourceFactory
- ☒ org.hawk.graph.updater.GraphModelUpdater
- ☐ org.hawk.modelio.xml.listeners.ModelioGraphChangeListener
- ☐ org.hawk.modelio.xml.metamodel.ModelioMetaModelResourceFactory
- ☐ org.hawk.modelio.xml.model.ModelioModelResourceFactory
- ☐ org.hawk.timeaware.graph.TimeAwareModelUpdater
- ☐ org.hawk.uml.metamodel.UMLMetaModelResourceFactory
- ☐ org.hawk.uml.model.UMLModelResourceFactory

Back-end:

Min/Max Delay:

We will go with a local, Neo4j-based, EMF-focused index.

Index created

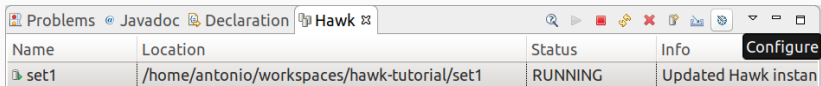


The screenshot shows an IDE window with a tab labeled 'Hawk'. Below the tab is a table with four columns: Name, Location, Status, and Info. The table contains one row with the following data:

Name	Location	Status	Info
set1	/home/antonio/workspaces/hawk-tutorial/set1	RUNNING	Updated Hawk instan

We have created an empty index. Time to configure it.

Configuring the index

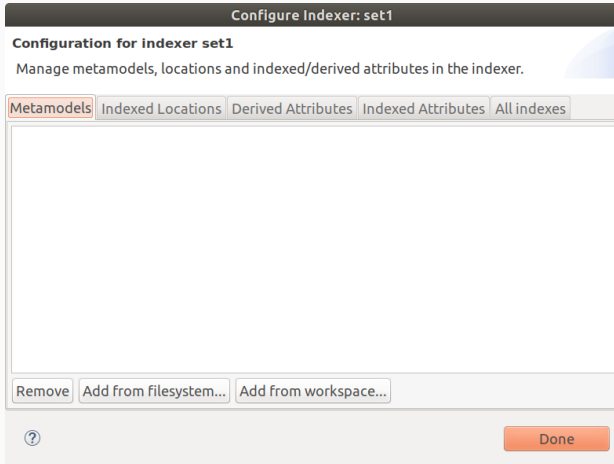


The screenshot shows the Hawk tool window in an IDE. The window has tabs for 'Problems', 'Javadoc', 'Declaration', and 'Hawk'. The 'Hawk' tab is active. Below the tabs is a toolbar with icons for search, play, stop, refresh, delete, and other actions. A 'Configure' button is visible in the top right corner of the table area. The table itself has four columns: 'Name', 'Location', 'Status', and 'Info'. There is one row with the name 'set1', location '/home/antonio/workspaces/hawk-tutorial/set1', status 'RUNNING', and info 'Updated Hawk instan'.

Name	Location	Status	Info	Configure
set1	/home/antonio/workspaces/hawk-tutorial/set1	RUNNING	Updated Hawk instan	

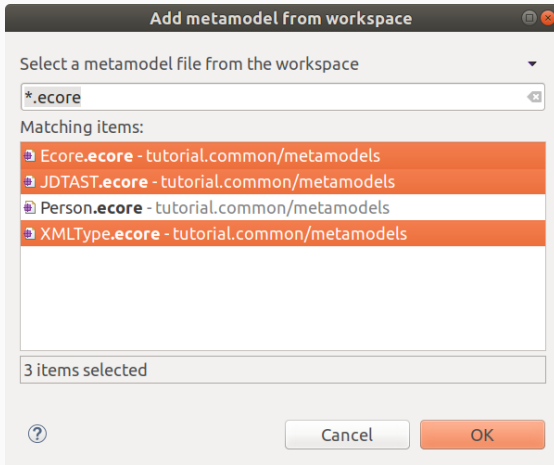
Let's click on the "Configure" button. This will open a new dialog.

Configure dialog



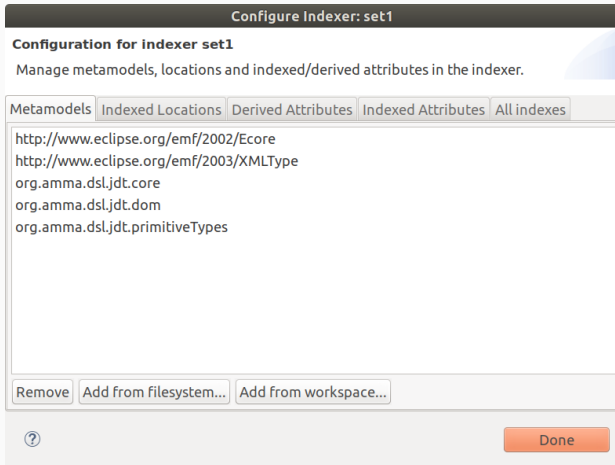
The dialog is divided into tabs for managing metamodels, locations to index, and indexed/derived properties.

Adding metamodels



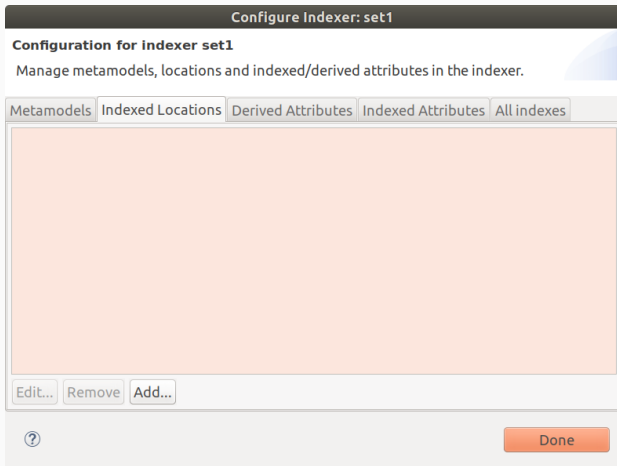
Click on “Add from workspace” and select the Ecore, XMLTypes and JDTAST metamodels. These will be turned into Neo4j nodes and edges.

Added metamodels



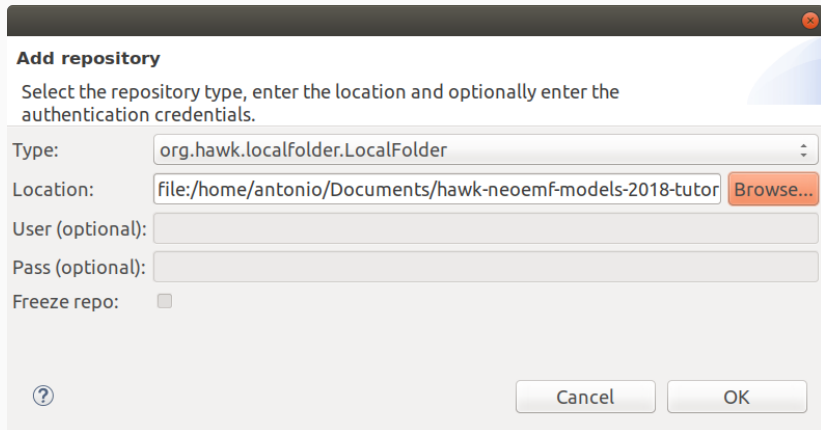
The metamodels have been registered: we can index models now.

Add locations



Go to the “Indexed Locations” tab and click on “Add”.

Adding a local folder



Add repository

Select the repository type, enter the location and optionally enter the authentication credentials.

Type:

Location:

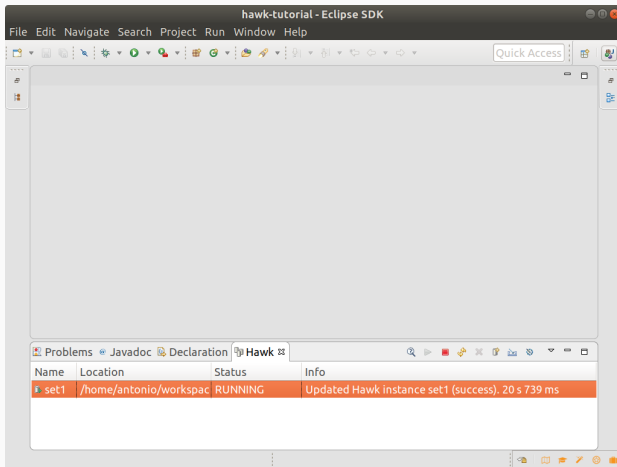
User (optional):

Pass (optional):

Freeze repo: ☐

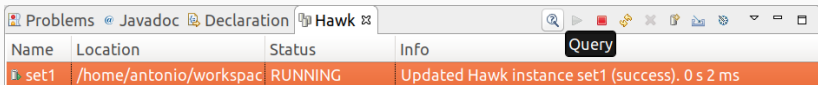
We want to index the “LocalFolder” in the
`tutorial.common/models/set1` folder.

Added local folder



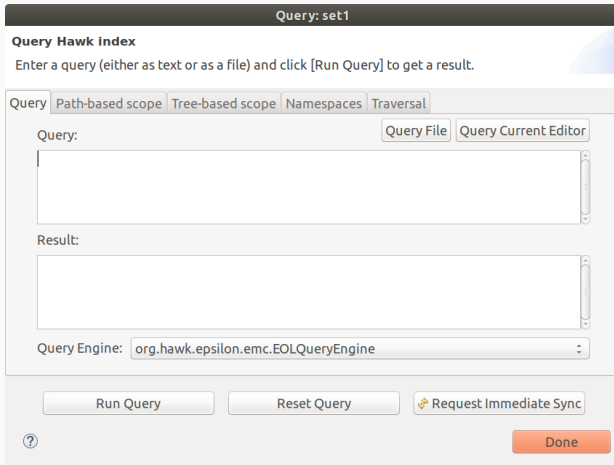
Wait a bit, watching the “Status” column, and the models will be indexed. Once it’s done, the index is ready to be queried.

Query button



Click on the “Query” button.

Query dialog: main tab



The screenshot shows a web-based interface for a query engine. At the top, a dark header bar contains the text "Query: set1". Below this, the main heading is "Query Hawk index". A sub-header instructs the user: "Enter a query (either as text or as a file) and click [Run Query] to get a result." Below the instruction are five tabs: "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Query" tab is currently selected. Inside this tab, there are two buttons: "Query File" and "Query Current Editor". Below these buttons is a large text input field for the query. Underneath the query field is a "Result:" label followed by a large text area for displaying results. At the bottom of the tab is a "Query Engine:" label and a dropdown menu showing "org.hawk.epsilon.emc.EOLQueryEngine". At the very bottom of the dialog are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (which has a small gear icon). In the bottom left corner is a help icon (a question mark in a circle), and in the bottom right corner is a red "Done" button.

The “Query” dialog is quite involved: let’s go bit by bit.

The main page allows us to enter the query by hand, from a file or from the current editor, run/stop/reset it, and request manual syncs.

Query dialog: path-based scope

The screenshot shows a window titled "Query: set1" with a subtitle "Query Hawk index". Below the subtitle is a instruction: "Enter a query (either as text or as a file) and click [Run Query] to get a result." There are five tabs: "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Path-based scope" tab is selected. It contains two text input fields: "Context Repositories (comma separated (partial) matches using * as wildcard):" and "Context Files (comma separated (partial) matches using * as wildcard):". Below these fields is a checkbox labeled "Start with files rather than types for Type.all (faster for small fragments in large graphs):" which is currently unchecked. At the bottom, there are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (with a refresh icon). A "Done" button is in the bottom right corner. A help icon (?) is in the bottom left corner.

We could optionally limit the scope of the query to specific repositories and/or files. This can be useful when querying large collections of models.

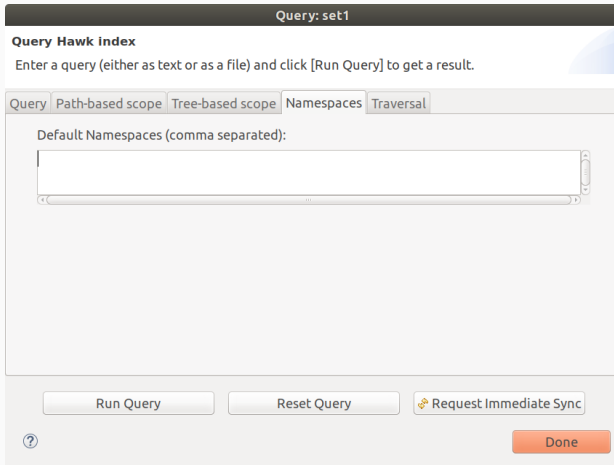
Patterns are essentially glob-style: e.g. `folder/f*.xml`.

Query dialog: subtree-based scope

The screenshot shows a software window titled "Query: set1". Inside, the "Query Hawk index" section has a text box for a query and a "Run Query" button. Below this is a tabbed interface with five tabs: "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Tree-based scope" tab is selected. It contains a text field labeled "Subtree root context (for fragmented models - path within repository):" with a scroll bar. Below the text field is a checkbox labeled "Use derived edges to speed up Type.all in subtree queries:". At the bottom of the dialog are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (with a refresh icon). A "Done" button is in the bottom right corner, and a help icon (?) is in the bottom left.

Another option is to limit scope to the objects contained within a certain top-level file. This can be useful when querying hierarchically fragmented models (e.g. those created with EMF-Splitter by Antonio Garmendia).

Query dialog: default namespaces



The screenshot shows a software window titled "Query: set1". Inside, the "Query Hawk index" section contains the instruction: "Enter a query (either as text or as a file) and click [Run Query] to get a result." Below this is a tabbed interface with five tabs: "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Namespaces" tab is currently selected. It features a text input field labeled "Default Namespaces (comma separated):". At the bottom of the window, there are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (which includes a refresh icon). A help icon (?) is located in the bottom left corner, and a "Done" button is in the bottom right corner.

In some cases, “Type.all” might be ambiguous as we may have multiple types with the same unqualified name. Here we can specify which metamodel URIs should take precedence.

Query dialog: traversal scoping

The screenshot shows a dialog box titled "Query: set1". Inside, there's a section "Query Hawk index" with the instruction "Enter a query (either as text or as a file) and click [Run Query] to get a result." Below this is a tabbed interface with five tabs: "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Traversal" tab is selected and highlighted with a red border. Under the "Traversal" tab, there is a checkbox labeled "Enable Full Traversal Scoping (may affect performance – only for scoped queries):" which is currently unchecked. At the bottom of the dialog, there are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (which has a small icon to its left). In the bottom right corner, there is an orange "Done" button. A small question mark icon is located in the bottom left corner of the dialog area.

When following references from an object, we may want to ignore the ones that leave the path-/subtree-based scope that we defined before.

Query dialog: entering a query

The screenshot shows a window titled "Query: set1". Inside, the "Query Hawk index" section has a subtitle "Enter a query (either as text or as a file) and click [Run Query] to get a result." Below this are five tabs: "Query" (selected), "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal". The "Query" tab contains a "Query:" label, a "Query File" button, and a "Query Current Editor" button. A text area below "Query:" contains the text "return TypeDeclaration.all.size;". Below the text area is a "Result:" label and a text area containing the red text "[query has been edited since last results]". At the bottom of the dialog is a "Query Engine:" dropdown menu showing "org.hawk.epsilon.emc.EOLQueryEngine". At the very bottom are three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (with a gear icon). A "Done" button is in the bottom right corner, and a help icon (?) is in the bottom left corner.

Query: set1

Query Hawk index

Enter a query (either as text or as a file) and click [Run Query] to get a result.

Query Path-based scope Tree-based scope Namespaces Traversal

Query: Query File Query Current Editor

return TypeDeclaration.all.size;

Result:

[query has been edited since last results]

Query Engine: org.hawk.epsilon.emc.EOLQueryEngine

Run Query Reset Query Request Immediate Sync

Done

Let's go back to the main tab and enter a simple query about the number of instances of a class. Click on "Run Query", which will temporarily change to "Stop Query".

Query dialog: query executed

The screenshot shows a window titled "Query: set1". Inside, the "Query Hawk index" section indicates the "Query completed in 0 s 218 ms". Below this are tabs for "Query", "Path-based scope", "Tree-based scope", "Namespaces", and "Traversal", with "Query" being the active tab. The "Query:" label is followed by a text area containing the query `return TypeDeclaration.all.size;` and two buttons: "Query File" and "Query Current Editor". Below the query is a "Result:" label followed by a text area displaying the value `40`. At the bottom of the main area is a "Query Engine:" label and a dropdown menu showing `org.hawk.epsilon.emc.EOLQueryEngine`. The bottom of the window features three buttons: "Run Query", "Reset Query", and "Request Immediate Sync" (with a refresh icon). In the bottom right corner is a red "Done" button, and in the bottom left corner is a help icon (a question mark in a circle).

The query finished running, and here we have the results.

Now that we know the basic use of Hawk, let's move to the more advanced features.

Indexed attributes

Finding a type by its name (`findByName.eol`)

```
return TypeDeclaration.all.selectOne(td |  
  td.name.identifier = 'IdInputFileObject');
```

This normally involves...

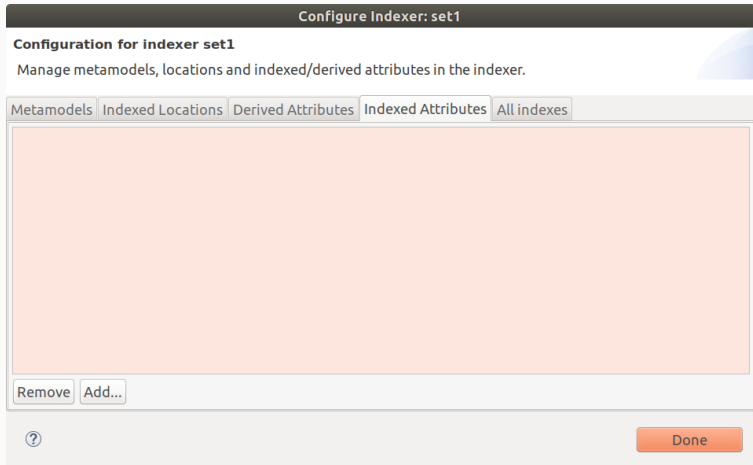
1. Iterating over all types
2. Following the “name” reference
3. Comparing the name

Replace with a lookup (`findByName-indexed.eol`)

We only need to tell it to index “SimpleName.identifier”:

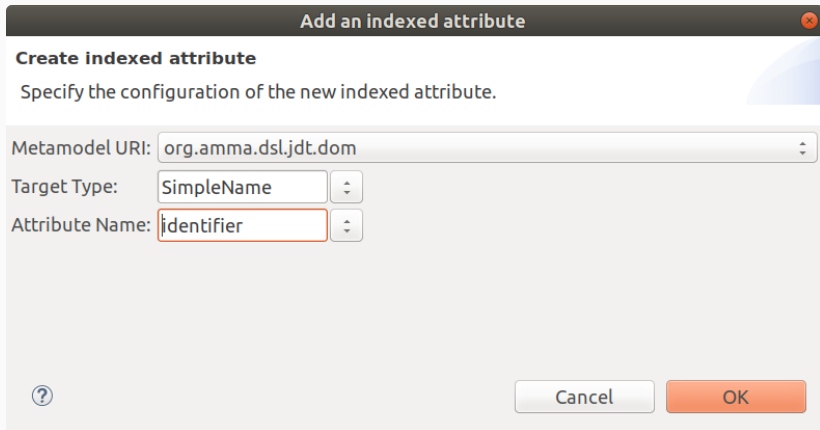
```
return SimpleName.all  
  .select(sn | sn.identifier='IdInputFileObject')  
  .eContainer.select(c|c.isKindOf(TypeDeclaration));
```

Indexed attributes tab



Go back to the “Configure” tab and select the “Indexed Attributes” tab.
Click on “Add”, and a new dialog should pop up.

Options for the indexed attribute



The screenshot shows a dialog box titled "Add an indexed attribute" with a close button in the top right corner. Below the title bar, the text "Create indexed attribute" is followed by the instruction "Specify the configuration of the new indexed attribute." The dialog contains three input fields: "Metamodel URI:" with the value "org.amma.dsl.jdt.dom", "Target Type:" with the value "SimpleName", and "Attribute Name:" with the value "identifier". Each field has a small dropdown arrow on its right side. At the bottom left is a help icon (a question mark in a circle). At the bottom right are two buttons: "Cancel" and "OK".

Add an indexed attribute


Create indexed attribute

Specify the configuration of the new indexed attribute.

Metamodel URI:

Target Type:

Attribute Name:



Use the combo boxes to pick the metamodel, type and attribute shown above. Click on "Add".

Derived attributes

Original query for finding singletons (`singletons.eol`)

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
  and md.returnType.name.fullyQualifiedName
    = td.name.fullyQualifiedName
  and md.modifiers.exists(mod:Modifier | mod.public = true)
  and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Can we do it faster?

- Checking if a method is public or static requires traversing references
- Same goes for checking if it returns an instance of itself
- In Hawk, **we can precompute this**
- When files change, only the affected values are recomputed

Use of derived attributes as precomputed values

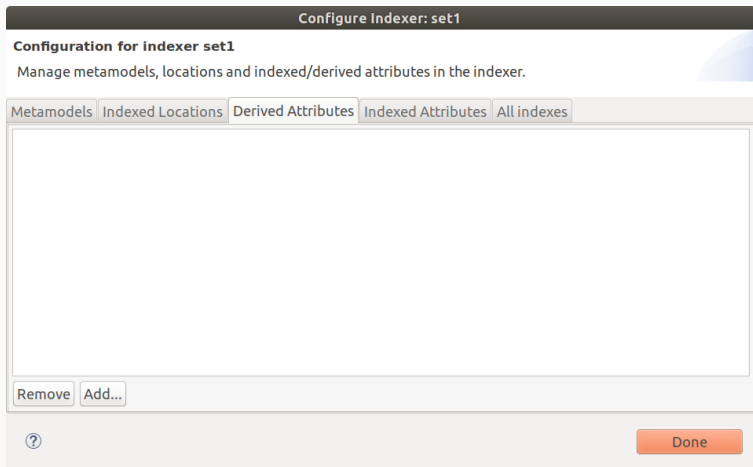
Original query (singletons.eol)

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
  and md.returnType.name.fullyQualifiedName
    = td.name.fullyQualifiedName
  and md.modifiers.exists(mod:Modifier | mod.public = true)
  and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Changed for MethodDeclaration derived attributes (singletons-dmethods.eol)

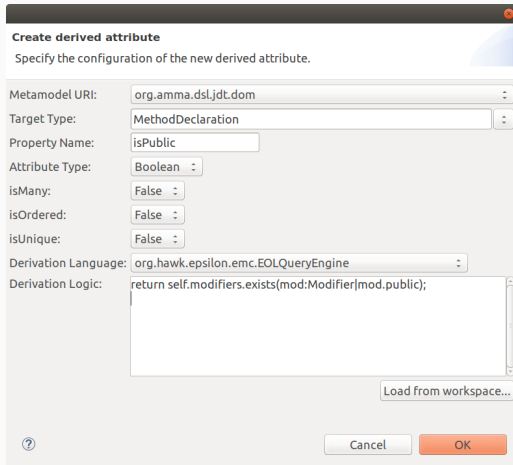
```
return TypeDeclaration.all.select(td |
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Derived attributes tab



Now we move to the “Derived Attributes” tab of the configure dialog.

Options for one of the derived attributes



Create derived attribute
Specify the configuration of the new derived attribute.

Metamodel URI:

Target Type:

Property Name:

Attribute Type:

isMany:

isOrdered:

isUnique:

Derivation Language:

Derivation Logic:

For the first derived attribute, fill in the form as above. You can load the EOL code straight from the workspace — no need to copy and paste!

Derived attributes are also indexed

Revised query (`singletons-dmethods.eol`)

```
return TypeDeclaration.all.select(td|  
  td.bodyDeclarations.exists(md:MethodDeclaration |  
    md.isPublic = true  
    and md.isStatic = true  
    and md.isSameReturnType = true));
```

Can we do it faster?

- Right now, we need to go through *all* type declarations and then filter by methods
- What if we go from the methods to the types instead?
- In Hawk, **top-level selects can replace iteration with lookups when using derived attributes**

Use of derived attributes as index keys

Previous query (singletons-dmethods.eol)

```
return TypeDeclaration.all.select(td |  
  td.bodyDeclarations.exists(md:MethodDeclaration |  
    md.isPublic = true  
    and md.isStatic = true  
    and md.isSameReturnType = true));
```

Revised to use index, with derived attrs. at top level
(singletons-dmethodsindexed.eol)

```
return MethodDeclaration.all.select(md |  
  md.isPublic = true and md.isStatic = true  
  and md.isSameReturnType = true  
)collect( td | td.eContainer ).asSet;
```

Flagging singletons directly

Previous query (singletons-dmethodsindexed.eol)

```
return MethodDeclaration.all.select(md |  
    md.isPublic = true and md.isStatic = true  
    and md.isSameReturnType = true  
).collect( td | td.eContainer ).asSet;
```

Can we do it faster?

- We could just flag types that are singletons
- This derived attribute might be less reusable, however

Final query for finding singletons

Previous query (singletons-dmethodsindexed.eol)

```
return MethodDeclaration.all.select(md |  
    md.isPublic = true and md.isStatic = true  
    and md.isSameReturnType = true  
).collect( td | td.eContainer ).asSet;
```

Final query (singletons-dtypes.eol)

```
return TypeDeclaration.all.select(td | td.isSingleton = true);
```

Hands-on time!

**This time, we will show how to use indexed
and derived attributes.**

Derived edges

Toy example: Person metamodel

- Person metamodel, with “parents” references.
- We want to be able to quickly find siblings, grandparents, uncles/aunts, cousins, second-cousins, ancestors...
- We can precompute this in Hawk with **derived edges**

Derivation logic for “grandparents” (Person_grandparents.eol)

We need a flat list and not a list of lists, so we use “flatten”:

```
return self.parents.parents.flatten;
```

Derivation logic for “siblings” (Person_siblings.eol)

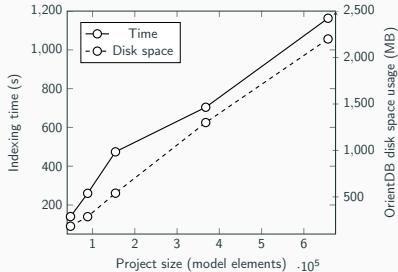
We can travel references in reverse with “revRefNav_name”:

```
return self.parents.revRefNav_parents.flatten.excluding(self);
```

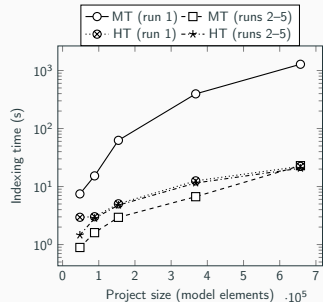
Last hands-on slot for Hawk.

We will show derived edges this time.

Hawk: integration into SOFTEAM Constellation [GDBK⁺16]



**Indexing times and index sizes
(OrientDB backend)**

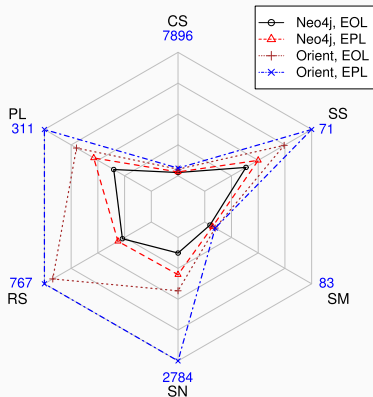


**Code generation times:
Modelio (MT), Hawk
(HT)**

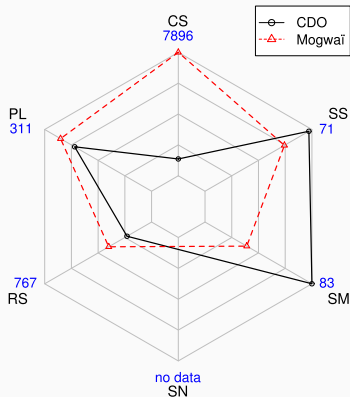
- Constellation: collaboration platform over Modelio models
- SOFTEAM needed search, couldn't change persistence
- Integrated Hawk as a library: initial indexing cost quickly paid off

Stress-testing remote query APIs [GDBK⁺17]

Hawk



Mogwai/CDO



- Included CDO, Hawk, Mogwai and ranged 1–64 clients
- Reverse reference navigation was crucial in having the SN Train Benchmark [SIRV17] query run quickly

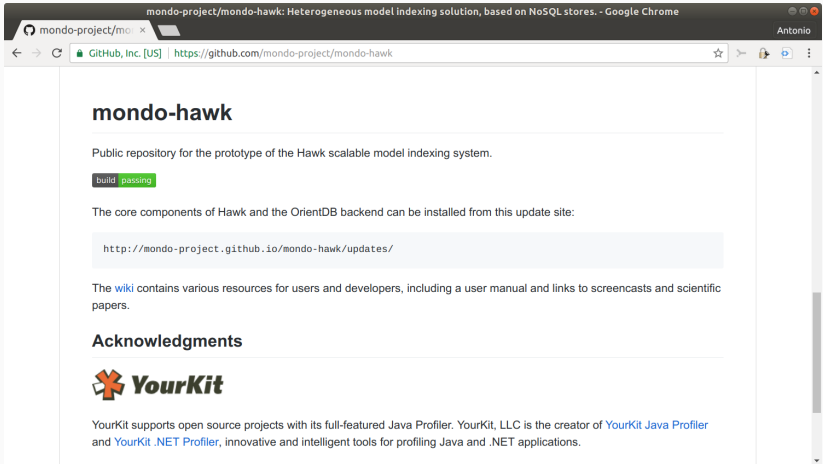
Time-aware queries in Hawk over versioned models [GDBP18]

```
var rs = RewardTableRow.latest.all.collect(row | row.getRewardShifts()).flatten();  
return Sequence { rs.min(), rs.max(), rs.average() };
```

```
operation RewardTableRow getRewardShifts(): Sequence {  
  var v = self.versions;  
  if (v.size <= 1) { return Sequence {}; }  
  else { return v.subList(0, v.size - 1).collect(v | v.value - v.prev.value); }  
}  
operation Sequence average() { return self.sum() / self.size(); }
```

- Extended Hawk with Greycat temporal graph support and time-aware query engine / updater components
- Can index entire Subversion-based history of a model, and ask things about its history through a new set of time-aware primitives
- Above query is finding descriptive stats for reward table shifts in a models@run.time system
- Presenting this work at 14:00 (MRT'18) — hope to see you there!

Hawk: project website



- <https://github.com/mondo-project/mondo-hawk>
- Recently accepted as Eclipse Incubator project (moving soon!)

Hawk: summing up

So far...

- Hawk is good for indexing an existing collection of model files
- You can efficiently answer queries from the index
- Indexed/derived features can be used to speed up queries

Ideas in the roadmap

- Extensible UI, covering differences in options for components
- Horizontal scaling (a flock of Hawks?)
- Web UI based on Thrift API
- More backends! (Triple stores? Neo4j 3.x? MapDB?)
- Better Git connector (JGit-based)
- Extending EOL with Linear Temporal Logic
- Visualizations based on time-aware queries



Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S Kolovos, Marcos Aurelio Almeida da Silva, Antonin Abherve, and Alessandra Bagnato.

Integration of a graph-based model indexer in commercial modelling tools.

In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 340–350, Saint Malo, France, 2016. ACM Press.



Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S. Kolovos, Ran Wei, and Richard F. Paige.

Stress-testing remote model querying APIs for relational and graph-based stores.

Software & Systems Modeling, pages 1–29, June 2017.



A. García-Domínguez, N. Bencomo, and Luis H. García Paucar.
Reflecting on the past and the present with temporal graph-based models.

In Proceedings of the 13th International Workshop on Models@run.time, 2018.

To be published.



Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró.
The Train Benchmark: cross-technology performance evaluation of continuous model queries.

Software & Systems Modeling, January 2017.